The Common Component Architecture (CCA)

Contact

- David Bernholdt, ORNL

Links

- CCA Forum web site (very poor at the moment, improvements underway)
- TASCS project web site (this is the large DOE SciDAC project that currently leads the development of the CCA)
    - Links to several overview presentations and the CCA tutorial web site, http://tascs-scidac.org/documents
- A long but comprehensive overview of the CCA project, http://eprints.cca-forum.org/5/ (if you can't access the paper request a copy from David Bernholdt)
- An eprint repository for CCA-related publications, http://eprints.cca-forum.org
- An iMesh CCA component for this project

Putting Things in Perspective

Opinion of David Bernholdt

There are many *domain-specific* frameworks that provide a modular environment that supports development of applications targeting a given scientific domain (with varying breadth). They typically embody (implicitly or explicitly) some assumptions about the workflow of computations in the target domain, and provide some basic utility functionality that is generally useful across the kinds of applications the framework was designed to support. They are very useful, but in my experience, as users try to push domain-specific frameworks further and further from their original domains, invariably limitations caused by assumptions embodied in the original design (either consciously or unconsciously) will make it increasingly challenging to implement the desired new capabilities. In this project, I would classify both SHARP and Salome as domain-specific frameworks.

The CCA is a *generic component architecture* specifically design for the needs of high-performance (primarily parallel, but also distributed) scientific computing. As a generic component architecture, it focuses at the very fundamental challenge of providing a methodology to encapsulate software modules (as *components*), characterizing their interactions with other software modules (though *interfaces* or *ports*), hooking them together, and executing the resulting application. In and of itself, a generic component architecture does not attempt to define computational workflows or utilities that serve a particular scientific domain. Some people may therefore perceive that a generic component architecture like the CCA does not add value, and it is true that in many cases where there is a large body of well-established code, there *may* not be much benefit to introducing CCA into the mix. However, when (a) developing new complex simulation environments, or (b) extending existing code bases into new areas, or (c) integrating or coupling simulations (within or especially across scientific domains), there is a good chance that introducing a generic component architecture can help make the software aspects of the problem more tractable. Consider that infrastructure supporting computations in a particular domain can be expressed as components (via a specification like the CCA), and layered on top of the fundamental capabilities of a generic component framework (such as defined by the CCA), and that because a generic component environment is agnostic with respect to domain and computational workflow, it allows you the flexibility to build up the supporting infrastructure you need for your applications. Most domain-specific frameworks can either be ported to sit on top of a generic environment, or can *interoperate* with generic component environments (so that components in the domain-specific framework can invoke components in the generic framework, and vice versa).